Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 16 (2013) 469 – 474

Procedia Computer Science

Conference on Systems Engineering Research (CSER'13)
Eds.: C.J.J. Paredis, C. Bishop, D. Bodner, Georgia Institute of Technology, Atlanta, GA, March 19-22, 2013.

Review of Agile Case Studies for Applicability to Aircraft Systems Integration

Robert Carlson[a,b,*], Dr Richard Turner[a]

[a] *Stevens Institute of Technology, School of Systems and Enterprises, Castle Point on Hudson, Hoboken, NJ, 07030, USA*
[b] *The Boeing Company, P.O. Box 3707, Seattle, WA 98124, USA*

Abstract

The current state of the art of aircraft systems integration is similar to the traditional serial software engineering waterfall method. Agile methods transformed software engineering with a focus on incremental iterative value added design. This paper reviews selected non software agile case studies for lessons that are applicable to implementing agile methods to transform the aircraft systems integration process.

Keywords: Agile Methods; Systems Integration

1. Introduction

In 1993, R. Belie noted that the aerospace industry faced challenges of a rapidly changing world coupled with increased competition and more demanding requirements which threatened to create unaffordable aerospace systems [1]. Mr. Belie contended that the industry had responded to these environmental changes by increasing team size and expanding development cycles which had the opposite effect of increasing costs and slowing innovation. Mr. Belie's proposition was to embrace integrated product development of small cross functional teams and embrace the use of computerized tools to increase the ability to iterate. Rapid iteration would facilitate change incorporation and enable the teams to explore a wider range of design options. Mr. Belie's proposal has similarities to agile methods with its focus on team based development concentrating on feature development and rapid iteration.

Software systems engineering was transformed by the introduction of agile methods. Agile methods were developed as a reaction to the traditional software waterfall development method where software was progressively developed in serial stages. When issues were uncovered, the software had to revert back to an earlier phase causing significant rework, schedule delays, and cost overruns. Agile methods are anchored on an iterative and incremental development cycle that quickly produces functional application versions that deliver value based solutions to the customer. Agile transformed software development so perhaps there are some useful methods that can be applied to

* R Carlson, robert.c.carlson@boeing.com.

doi:10.1016/j.procs.2013.01.049

aircraft systems integration.

Commercial aircraft product development has high barriers to entry due to the cost, long development cycle, and extensive certification requirements. Product development costs for new commercial aircraft are 5+ billion dollars [2] which means a company literally bets its future on each development effort. Recent history has seen significant cost and schedule overruns for new aircraft designs. Why? The pace of technological evolution has increased, yet aircraft design relies on increasing team sizes and extending the design cycle to deal with increased requirements and complexity. To make matters worse, we throw technological solutions at the problem hoping to cure it. Alexander Hellemans notes that the A380 delays were ostensibly caused by an incompatibility of the CAD systems between the different Airbus design divisions [3]. Aircraft systems are increasing in number and complexity which exacerbates the systems integration challenge. As technology evolves, each system requires greater development effort and more attention to how the system integrates with other systems on the aircraft. Interface standards are essential and the underlying architecture of how the systems interact becomes crucial. Architectural changes become increasingly expensive as development progresses. Systems integration is a critical skill to keep these large scale projects on track. However, aircraft systems integration is still based on a serial development methodology that seeks to address complexity with ever larger teams. The larger teams require more communication and the technological complexity slows the development pace. Furthermore, the serial process delays integration until late in development which results in extensive rework, escalating costs, and schedule slippage as issues are uncovered.

This paper evaluates several case studies of using agile methods in non software industries to identify lessons that are essential to implementing agile methods on aircraft systems integration. Case studies were selected to show how to address scalability and draw attention to the need for organizational paradigm shifts as well as the need for a foundational architecture and robust change management. The Johns Hopkins CubeSat case demonstrates the application of a highly iterative agile process to a complex engineering problem and the resulting benefits in decreased cycle time and cost as well as increased innovation. The Nokia example describes how agile methods were deployed in a large organization, how an underlying architecture was used to preserve component reusability across product lines, and how configuration control was accomplished by using a high level scrum team. The third case discusses Ipek Ozkaya's proposal for how to organize agile teams with either a feature driven or component based focus and the need for flexible organizational structures. The DOD procurement cases show how agile methods were used to reduce acquisition cycle time and reveal more organizational lessons. The Sonova case shows an innovative approach to managing agile product development with a large number of requirements and limited resources by creating a prioritization method to allocate the scarce systems engineering resources.

Nomenclature

API = Application Programming Interface
CAD = Computer Aided Design
DOD = Department of Defense
JHU/APL = Johns Hopkins University / Applied Physics Laboratory
JOPES = Joint Operational Planning and Execution System
MMBD = Multi-Mission Bus Demonstrator
RE = Requirements Engineering

2. Discussion

On the surface, software engineering and aircraft systems integration seem to have little in common. However, the foundation of both lies in a systems level approach to integrating the entire system. This requires an overarching strategy about architecting the systems, allocating the requirements, and modularizing the systems into manageable work packages with crisply defined interfaces. Aircraft systems design is still very much like the legacy software waterfall method with serial phases of conceptualizing, designing, testing, and verifying and validating. Aircraft design shares many similarities with large software development projects such as a multi-year development cycle with evolving requirements, configuration control requirements, and interface challenges. Aircraft systems integration is similar to software engineering and competitive advantage can be created by implementing agile methods to transform aircraft systems integration. The real benefit from implementing agile methods is controlled

incremental iteration that progressively evolves the product with the focus on adding value to the customer. This paper reviews selected agile case studies from non software implementations to identify key lessons.

2.1. Johns Hopkins Multi-Mission Bus Demonstrator (MMBD) CubeSat [4]

The CubeSat program was established to enable universities to create small satellites with standardized sizes and masses so they could be inexpensively launched as piggyback missions on other space launches. The Johns Hopkins University Applied Physics Laboratory (JHU/APL) used agile methods to develop their Multi-Mission Bus Demonstrator CubeSat. JHU/APL considered their MMBD project super-high-tech because it used new, non-proven concepts and system components such as the bus structure and the solar array deployment mechanism while meeting NASA's high standards of quality assurance and reliability. JHU/APL estimates they were able to develop their CubeSat for about 3% of the cost ($10M) and 25% of the schedule (14 months) of a typical satellite.

This case is an example of development of a complex system using agile methods. JHU/APL flattened the development organization structure to two levels with functional leads from Payloads, Electrical, Software, Mechanical, and Avionics reporting to a single program manager. The program manager was granted authority to institute any changes as long as the change process was followed and the overall project cost and schedule were not compromised. The teams were co-located which reduced the number of meetings to disseminate information and enabled a non linear change process. JHU/APL used incremental test methods which favored testing each component and deferred integration tests till late in the development process. This allowed problems to be identified and corrected early and emphasized incremental creation of a working system.

Key lessons from Johns Hopkins MMBD CubeSat are that empowered co-located teams enable design flexibility. Incremental testing effectively increases the iteration pace which enables issues to be revealed and dispatched. A strong change process is essential. The flat organization and resulting flexibility with the emphasis on producing functioning articles played a crucial role in Johns Hopkins CubeSat success.

2.2. Nokia Agile Requirements Development [5]

Nokia sought to decrease their development time and increase their capability to react to changing customer requirements. They implemented scrum as the development method for user requirements in two organizations containing over 500 phone software developers. Scrum is the predominant agile method where development iteration is time boxed into a 30 day sprint. The team manages a backlog file that prioritizes the current requirements and the team meets daily to update each other on progress and allocate remaining tasks.

One challenge Nokia faced was scale. It was not practical to have a team of 500, so they created a master backlog file and allocated the master backlog to team backlog files. They had to create a higher level scrum team to manage the master backlog file and allocate the requirements. Another issue they had with scale was that decisions were not clearly documented and communicated to other design teams. Architectural changes were not quickly flowed down to impacted teams. The example given is that battery life had to be increased which meant decreasing power consumption which took a while to percolate downward.

Another issue was that Nokia used "frameworks" to generalize the behavior of their product lines. The framework does not directly fulfill user requirements; it interacts with applications that fulfill the requirements. Nokia calls these derived requirements. The issue was that the frameworks varied in their functional scope and how closely the framework operated to the user experience. Each framework had different product lines it supported and changing a framework required an in depth analysis of how the changes to the framework might impact all of the other product line members. The complexity of the framework implementation required more experienced and senior level persons because they had to perform the in depth analysis of the abstracted requirements and assess the ramifications of changing the framework.

Key lessons from Nokia's experience are that agile can scale, but there must be a higher level scrum orchestrating the work distribution through a master backlog file which can create bottlenecks and require schedule integration and management. Architectural decisions must be prioritized, documented, and flowed down so work is not invested in areas which subsequently change. More experienced and highly skilled persons are needed because requirements have to be decomposed into physical and logical technical requirements that require technical competence and awareness of the overall architecture.

2.3. Ozkaya Agile Development and Architecture [6]

Ipek Ozkaya proposed an agile process that invests in the architecture first because she contends that the architecture can be used to satisfy the quality requirements and the functional requirements can be allocated to the architectural elements. Furthermore, the critical architectural decisions are made earlier and clearly documented and architectural issues can be easily identified and managed. Many agile implementations organize teams around features and the team is responsible for changing anything, anywhere to implement the new feature. Other agile implementations organize teams around components (E.G. presentation layer, domain layer, driver layer) and the teams act as component owners who need to be contracted with to effect change on the component. Feature driven organization runs the risk of having many teams enact simultaneous change which could destabilize the system. Component based organizations create silos of ownership and inhibit shared ownership. Neither organizational structure seems agile as project scale increases.

Ozkaya proposes to organize around matrix teams that use an established architecture. The system is decomposed into architectural layers (E.G. presentation layer, domain layer, data access layer). Each architectural layer is a common service that gets called through its API. This approach is similar to the component owner agile team structure, except only the API is frozen and anyone is free to change the component as long as the API is stable. API changes must be evaluated, configuration controlled, and regression tested. The features are allocated to the architectural layers while trying to minimize dependencies across the architectural layers. Features are manifested through the API. A matrix of teams is created and led by a scrum of scrums who orchestrate the effort. Every scrum team has a member from each of the architectural layers who acts as the team's representative to implement features. This matrix team model provides for shared knowledge and ownership and seems to be scalable. Initially, some temporary scrums need to be organized to build the architectural layers and create the architectural layer team member knowledge base. Once the architecture is implemented, the temporary scrums are disbanded and the people become architectural representatives in the feature driven scrums.

Key lessons are to create the architecture early and build the architectural knowledge base. This forces the architectural decisions to be made early and avoids an unstable architecture that could cause re-work later as architectural issues surface. Dynamic teams are an interesting concept that prevents specialization and promotes a shared vision. It also enables teams to be deconstructed and reconstituted to better fit the next iteration's feature workload. The concept of treating the architecture as a common service accessible through an API provides abstraction from the implementation details and forces everyone to work to defined interfaces.

2.4. Implementing Agile for DOD IT Acquisition [7]

Benito and Casagni presented two cases of DOD IT acquisition projects that used agile methods to refine application requirements as they procured custom web applications. Program "A" was a case study implementing a collaborative, web-based tool for sharing information and services. This project had a government and a vendor team of about five individuals each. The high level requirements were decomposed into features which were decomposed into user stories (use cases). The high level requirements were traced to the features and this matrix was used for testing and customer sign off. Features were targeted for incremental development through 4 to 6 iterations composing a release that was scheduled every 6 months.

Key lessons were that security needs must be architected early to avoid re-work. There were also problems with testing because the users had not defined the verification tests up front. Some interesting lessons were that agile is a continual improvement process and mistakes needed to be planned for by reserving the iteration before the release for cleanup. A recommendation was to keep the team size to five or less.

A second DOD project to use agile methods was JOPES (Joint Operational Planning and Execution System). The typical DOD acquisition release was 18 to 36 months, and they sought to reduce that to a 30 to 90 day cycle. There was a team of 5 to 20 software developers. A key lesson from this implementation was that it was important to have an agile coach who educated the people on how the process should work. Initially, programmers were reluctant to expose their code and feared losing their status as lead engineers. Another issue was that testing was done by an independent department and took months to gain fielding approval. This problem was addressed by inviting testers into the team so they played an integral role from the start.

2.5. There's Never Enough Time: Doing requirements under resource constraints, and what Requirements Engineering can learn from agile development [8]

Bernd Waldmann presented the case study of the Sonova Group who manufactured hearing solutions. Waldmann notes that projects often start late and never have enough resources to implement everything, so he relates how agile methods were used to segment a Sonova project into manageable chunks that focused on creating product value. The project was to create the fourth generation of a hearing instruments platform that created reusable components that could be used across multiple brand names, price points, and power classes. The project took over 3 years and engaged over 100 R&D engineers resulting in over 1200 system level requirements. The customers include component manufacturers and software engineers.

The project began with what they called their triage strategy. The first step was an evaluation to identify the areas where requirements engineering (RE) added the most value. Project areas that were new or involved multiple products were deemed areas where RE could add the most value. Interestingly, new areas that could be prototyped were demoted from valuable because prototyping produced a tangible object that could be better evaluated than a pile of paper requirements. The remaining candidate areas where RE could add value were filtered against the business risk of having poor requirements. The triage strategy helped the requirements engineers determine which areas of the project would benefit most from their expertise.

Another adopted agile method was iterative development. The group determined that there were long lead items, and focused on developing those requirements with the first planned release at 6 months. The second release focused on high level architectural requirements and requirements that impacted multiple products because it was important to lock in the architecture to minimize rework. Periodic releases followed about every 4 months which defined the features required for each specific product. The team employed a similar value based triage strategy to determine which products and features needed to be worked earliest. A final release was scheduled to reverse engineer verification requirements for the areas that had been demoted from high value added.

This was a novel application of agile methods to a requirements engineering project. The author relates how the work effort was prioritized to focus work effort on the place where the most value could be created. One lesson was to inform the stakeholders about the process and triage strategy because the stakeholders were used to the traditional waterfall method. Another lesson was that 4 months between releases was too long because the requirements engineers did not have enough feedback from the customers. The recommendation is to deliver more releases with fewer features to accelerate the feedback/reaction cycle.

3. Conclusions and Future Work

Agile methods have proven their worth in software engineering and there may be opportunity to implement some of the concepts in non software implementations. It is worthwhile to examine agile implementations to learn what worked and what did not. Aircraft systems integration has a long cycle time and is expensive so implementing agile methodologies into aircraft systems integration could create a competitive advantage.

Most of these cases ran into scaling issues. Nokia's response was to create a lead scrum team with overall responsibility and create teams that were feature focused. Ozkaya used a lead team and also proposed to create a dynamic matrix of teams that first focused on creating the architecture and later became feature oriented with team members fluent in each of the architectural layers. Key lessons are to plan for scaling and use some form of master team to orchestrate sub teams and manage the master backlog and feature allocation. A rigorous configuration control process must be followed to keep everyone on the same page. Diverse teams with knowledgeable members from all areas are needed to encourage shared vision and accountability. Agile methods work best with smaller teams, so it may be best to change the organizational structure as was done with the Johns Hopkins CubeSat. There are clearly cultural issues that need to be addressed to create an effective agile implementation.

The DOD cases noted the importance of having an Agile coach to educate the teams on how the process was supposed to work. The Sonova example had a similar lesson in that the stakeholders needed to be educated about the process and the triage strategy for prioritizing work. Co-location enabled the Johns Hopkins teams to facilitate rapid design iteration which enabled flexibility and innovation. It was interesting to note that the Sonova example found 4 month iterations were too long and stifled the feedback cycle. Process training sounds like a worthwhile investment to change the culture, create appropriate expectations, and quickly implement effective methods.

The Nokia and Sonova examples emphasized the need for prioritized work to support long lead efforts and early architecture decisions. The Johns Hopkins CubeSat employed incremental testing to keep the teams focused on meaningful work. The DOD examples ran into architectural issues related to security and testing which demonstrates the need to prioritize the work to reduce wasted effort and rework. One lesson is to formulate the architecture early and document subsequent changes. Another lesson is to use API interfaces to insulate the architectural layers. The Sonova example showed how prioritization could be used to focus the work on the areas where the most value could be created. It was interesting that some work was targeted for prototyping to reduce the work load on critical areas and get quickly to a tangible product that would provide the best feedback.

The challenges to aircraft systems integration are that systems are increasing in complexity and dependency while the systems integration process remains serial and attempts to deal with increased complexity through larger team sizes and extended development time. This legacy approach increases costs and development time due to the increased communication requirements and delays integration till very late which causes extensive rework and schedule impact. S. VanderLeest and A. Buter reveal that the success of agile software projects is related to having an architecture that allows independence of the teams and getting multiple early releases to the customers which provides feedback to the development teams [9]. VanderLeest and Buter contend that requirements changes are inevitable and the waterfall method delays integration till late in development. One of the key benefits of agile methods is that they use rapid iteration to focus development on producing functioning objects. Lessons are that aircraft systems integration should focus on creating the underlying systems architecture and rigorously process controlling API (interface) changes. Rapid iteration and incremental testing should be embraced to reveal deficiencies early. Organization structures will have to change to create empowered cross functional teams with responsibility for achieving features or components. Process training and Agile coaches would be helpful to guide people through the transition and show them where they fit in the new paradigm.

Employing agile methods for aircraft systems integration promises great flexibility and innovation while reducing cost and schedule. However, this must be tempered by the fact that aircraft development requires a high standard of safety. Case studies of agile implementations of mission critical systems should be examined, and small systems integration prototypes similar to the Johns Hopkins CubeSat are essential to validate the application of agile methods to aircraft systems integration. Lean methods would be a good complement because they are more process based than agile methods and aircraft development is very much process based to comply with certification requirements. Boehm and Turner [10] conclude that staffing, culture, and communications are important success factors for agile implementations. Future research should examine the environmental factors prevalent in the aircraft design industry to evaluate their suitability for nurturing agile methods.

References

1. R. Belie, Three Enabling Technologies for Integrated Product Development, AIAA 93-0923, (1993).

2. A350 Residual Calculations on Hold as Airbus Ponders Future, Aviation Today, (April 17, 2006).

3. A. Hellemans, Manufacturing Mayday, IEEE Spectrum, (2007).

4. P. Huang, A. Darrin, A. Knuth, Agile Hardware and Software System Engineering for Innovation, IEEE Aerospace Conference, 10.1109/AERO.2012.6187425 (2012) 1-10.

5. J. Kuusela, J. Savolainen, A. Vilavaara, Transition to Agile Development: Rediscovery of important requiremetns engineering practices, Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, 10.11009/RE.2010.41 (2010) 289.

6. I. Ozkaya, Agile Development and Architecture: Understanding scale and risk, Webinar, January 2012.

7. R. Benito, M. Casagni, K. Mayfield, C. Northern, Handbook for Implementing Agile in Department of Defense Information Technology Acquisition, MTR100489 Technical Report, 11-0401, (2010) 8-38.

8. B Waldmann, There's Never Enough Time: Doing requirements under resource constraints, and what Requirements Engineering can learn from agile development, International Requirements Engineering Conference (2011).

9. S. VanderLeest, A. Buter, Escape the Waterfall: Agile for Aerospace, Digital Avionics Systems Conference, 10.1109/DASC.2009.5347438 (2009) 6.D.3-1-6 – 6.D.3-16.

10. B. Boehm and R. Turner, Balancing Agility and Discipline: a guide for the perplexed, Boston, 2004.